2014 47th Annual IEEE/ACM International Symposium on Microarchitecture

Wormhole: Wisely Predicting Multidimensional Branches

Jorge Albericio, Joshua San Miguel, Natalie Enright Jerger, and Andreas Moshovos
Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada
Email: {jorge, sanmigu2, enright, moshovos}@eecg.toronto.edu

***Abstract*—Improving branch prediction accuracy is essential in enabling high-performance processors to find more concurrency and to improve energy efficiency by reducing wrong path instruction execution, a paramount concern in today's power-constrained computing landscape. Branch prediction traditionally considers past branch outcomes as a linear, continuous bit stream through which it searches for patterns and correlations. The state-of-the-art TAGE predictor and its variants follow this approach while varying the length of the global history fragments they consider.**

This work identifies a construct, inherent to several applications that challenges existing, linear history based branch prediction strategies. It finds that applications have branches that exhibit *multi-dimensional* correlations. These are branches with the following two attributes: 1) they are enclosed within nested loops, and 2) they exhibit correlation across iterations of the outer loops. Folding the branch history and interpreting it as a multidimensional piece of information, exposes these cross-iteration correlations allowing predictors to search for more complex correlations in the history space with lower cost. We present *wormhole*, a new side-predictor that exploits these multidimensional histories. Wormhole is integrated alongside ISL-TAGE and leverages information from its existing side-predictors. Experiments show that the wormhole predictor improves accuracy more than existing side-predictors, some of which are commercially available, with a similar hardware cost. Considering 40 diverse application traces, the wormhole predictor reduces MPKI by an average of 2.53% and 3.15% on top of 4KB and 32KB ISL-TAGE predictors respectively. When considering the top four workloads that exhibit multi-dimensional history correlations, Wormhole achieves 22% and 20% MPKI average reductions over 4KB and 32KB ISL-TAGE.

I. Introduction

Branch prediction has long been a cornerstone of high-performance microarchitectures; even modest improvements in branch prediction accuracy can reap large performance improvements in modern processors [17]. Originally branch prediction was driven by the need for higher performance in deeply pipelined processors with large instruction windows. Today, branch prediction also serves to greatly reduce energy waste from wrong path instructions, a major concern in today's power-limited designs. Starting from Smith's work on branch prediction [24], several innovations including Yeh and Patt's work on pattern-based prediction [27] have greatly boosted branch prediction accuracy. Significant research activity in the 1990s increased predictor sophistication [4], [5], [13], [14], [16], [25]. More recently, additional improvements have been spurred by the branch prediction championship workshops [2], [8], [10], [12], [15], [18], [20], [22].

State-of-the-art branch predictors such as TAGE [21] and Perceptron [11], achieve high accuracy by intelligently identifying highly-correlated patterns in the branch direction stream, or *branch history* as it is commonly referred to. TAGE targets continuous patterns of varying length while Perceptron can, in principle, correlate with discontinuous patterns. Barring a major breakthrough in the way branches are predicted and given the accuracy achieved with state-of-the-art predictors, a fruitful path forward to further improve branch prediction accuracy is by targeting more specialized behaviors with small side-predictors [7], [19], [21]. One of the best predictors to-date, ISL-TAGE follows this approach by incorporating a loop predictor and a statistical corrector, each of which targets special cases.

This work identifies a branch direction pattern that a state-of-the-art predictor is practically unable to accurately capture and advocates devoting a small amount of extra hardware to boost accuracy. Although the specific pattern does not exist in all applications, the modest hardware cost coupled with the large gains in the applications where the pattern appears justify the introduction of this *wormhole* side-predictor.

Specifically, the wormhole side-predictor targets certain hard-to-predict branches that appear within nested loops. Targeting branches within nested loops is worthwhile since many applications spend considerable execution time in such loops. This work observes that these loops often contain branches whose direction stream is correlated with outcomes from previous iterations of the outer loop rather than recent outcomes of the inner loop. The direction stream of such a branch appears irregular and hard-to-predict when viewed as a continuous stream. However, *folding* the direction stream at the granularity of the outer loop reveals the existing strong correlations. To capture this behavior in a traditional manner, large local histories and many entries would be required.

This cross-iteration behavior motivates a rethinking of how branch predictors manage local histories: We propose representing histories as multidimensional matrices instead of linear vectors. We propose the *wormhole* branch predictor which is able to uniquely capture patterns in a multidi-

1072-4451/14 $31.00 © 2014 IEEE
DOI 10.1109/MICRO.2014.40

mensional local history space. Built on top of an ISL-TAGE baseline, the wormhole-enhanced WISL-TAGE[1] can better predict branches that exhibit multidimensional history correlations.

In summary, this work makes the following contributions: (1) It identifies a branch behavior inherent to several applications whose regularity can be exposed when branch outcomes are viewed as multi-dimensional matrices. (2) It proposes a novel low-cost specialized branch side-predictor that can accurately capture this behavior. Experiments demonstrate that for a modest 3.29% and 4.4% increase in hardware relative to a baseline 4KB and 32KB ISL-TAGE branch predictor, the proposed WISL-TAGE predictor can effectively predict these branches. For the subset of workloads studied (4 out of 40 applications) with the targeted behavior, WISL-TAGE reduces MPKI by 22% and 20% over the 4KB and 32KB ISL-TAGE baseline, respectively. Across all applications, the average MPKI reductions are 2.53% and 3.15%, respectively.

The rest of the paper is organized as follows. Section II discusses the application behavior that motivates the use of multidimensional histories for branch prediction. Section III explains the wormhole prediction concept using examples. Section IV reviews the baseline ISL-TAGE predictor while Section V presents the wormhole predictor. Section VI presents the experimental evaluation. Finally, Section VII concludes the paper.

II. Motivation

The wormhole predictor targets branches in inner loops whose direction stream is strongly correlated across iterations of the outer loop. The example pseudo code of Figure 1(a) shows such a branch, Branch 1 (*B1*). The code scans repeatedly over a set of objects in a 3D scene using the induction variable *j*. For each object, it uses the distance from the current position *p* to the positions of the objects stored in array *X* to decide whether further processing should take place per object. The direction stream of B1 is data dependent.

Branch predictors have traditionally considered history as a one-dimensional string of bits, where each bit represents the outcome of a previous instance of a branch. The branch predictor searches for repeating patterns among these history bits. Accordingly, existing branch predictors would try to find correlations within the continuous stream of B1 outcomes. Whether such correlations exist depends on the actual distance values. Any correlations so found would be the result of happenstance. However, as long as the distance values do not change much so as to exceed the specific threshold, the directions B1 will follow will be identical to those last time the inner loop executed in full.

[1]Pronounced "Wisely" TAGE.

Figure 1(b) shows an example direction stream for B1 where *1* and *0* represent taken and not taken branch outcomes. The direction stream is folded over a matrix such that each row represents an iteration of the outer loop, and each square of a row represents one iteration of the inner loop. As the example shows, B1 is highly predictable in terms of the outer loop iterations (columns). Thus, a mechanism that considers the branch history in a multidimensional fashion would successfully predict B1. Moreover, depending on the pattern, it will do so with fewer resources than a conventional predictor working over the continuous history.

B1 is hard to predict for existing global history-based predictors. Specifically, a global history-based predictor will have to contend not only with B1 but also with the intervening branches as well. Unless the intervening branches exhibit regular behavior, there will be numerous history patterns that would need to be observed. These will contain a large fraction of *don't care* values (values irrelevant to B1's outcome) obscuring the underlying pattern, making training difficult and time consuming. Would this situation improve if only B1's local history was considered? The answer is not necessarily. If B1's behavior is irregular within the loop, it would foil a typical local history based predictor using a limited amount of resources. Such a predictor would have to record a sufficiently long history pattern for every iteration of the inner loop.

A comparison of two local-history based predictors, one using a traditional uni-dimensional (1D) history and another folding history into a two-dimensional (2D) matrix can illustrate the difficulties of existing branch predictors and the potential of history folding. For simplicity the number of iterations of the inner-loop is assumed to be 100 and the predictors are allowed to track an unlimited number of branches. Each branch is limited to 2^{16} entries, so when necessary the 1D predictor *xor-folds* the pattern into a 16-bit index per branch. The misprediction ratio for B1 is 1% for the 2D predictor, while it is 15% and 14% for the 1D predictor that uses 8 and 64 bits of history respectively. It is not until 128 history bits are used that the 1D predictor accuracy improves with the misprediction ratio dropping to 2%.

In the example in Figure 1(a), B1's behavior is identical across iterations of the outer loop. The example in Figure 1(c) demonstrates that further opportunities exist for correlation even if the target branch's behavior is not identical across iterations of the outer loops. Figure 1(c) shows the calculation of one step of the Jacobi algorithm [3]. Focusing on Branch 2 (B2) of this algorithm, Figure 1(d) shows an example of a typical direction stream for B2. Its local history forms a diagonal pattern across the two-dimensional local history space. When the size of the matrix is large, the number of mispredictions for B2 using a traditional, local history based predictor is O(N), where N^2 is the size of the matrix. Using a predictor with a two-

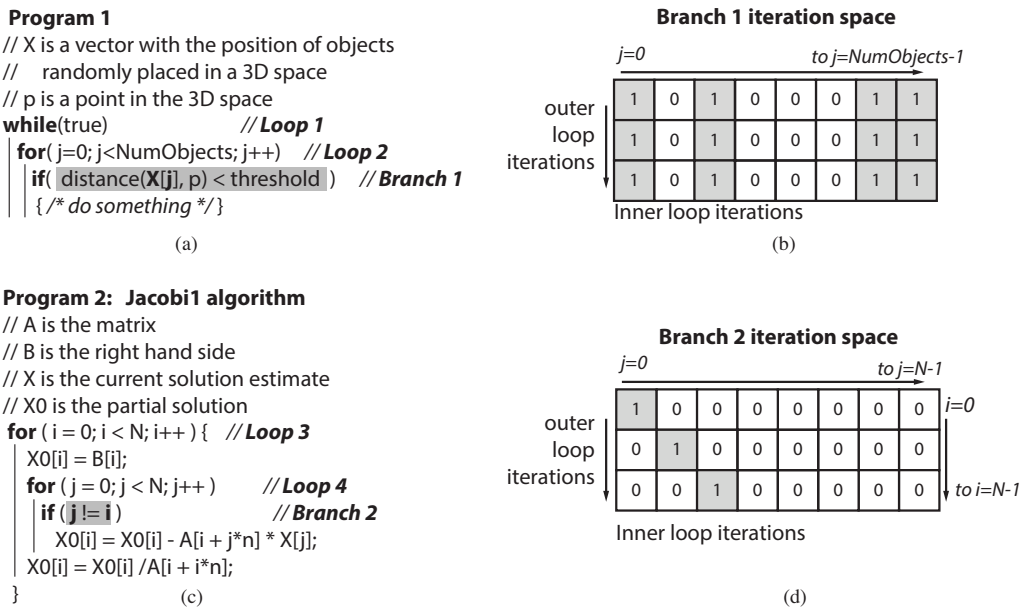


Figure 1: Example programs. a) Program 1; b) Outcome of *Branch 1* in *Program* 1; c) Program 2; d) Outcome of *Branch 1* in *Program 2*.

dimensional, folded history reveals the diagonal which can be predicted accurately (as Section III explains), thereby reducing the number of mispredictions to O(1).

These are only two simple examples where traditional branch predictors that consider one-dimensional branch histories would either fail or would need a large number of entries and resources to accurately predict them. However, folding these histories at appropriate points yields a multi-dimensional history that readily exposes the inherent repeating patterns.

III. Wormhole Concept

In this section, we describe the wormhole predictor concept by means of examples. For the purposes of this discussion assume that: 1) somehow we have identified a branch inside an inner loop that is a candidate for wormhole prediction, and 2) we can predict with high confidence how many times this inner loop will iterate (LP_{TOTAL}), and 3) we know how many iterations we have currently seen (LP_{CURR}). As the next section explains, ISL-TAGE can be extended to provide this information with little cost.

Let us first consider the example in Figures 1(a) and 1(b) where the target branch, B1, exhibits identical behavior every time the inner loop executes. Once such a branch is identified, predicting it is straightforward. During the first execution of the inner loop we record the directions of the branch yielding a vector of LP_{TOTAL} bits. This recording stops once the loop exits, that is when $LP_{CURR} = LP_{TOTAL}$. Next time we encounter the branch, we simply replay the recorded history using LP_{CURR} as the index.

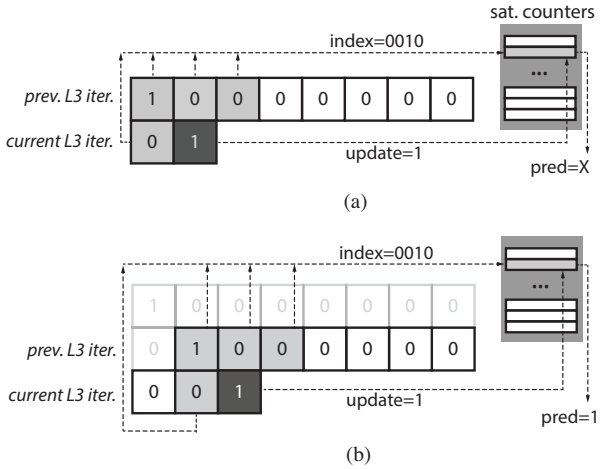


Figure 2: Diagonals Example.

While simple, this prediction method is appropriate only when the branch exhibits almost identical behavior every time the inner loop executes. In practice, we found this predictor unsatisfactory.

Let us next consider the example in Figures 1(c) and 1(d) where the target branch, B2, exhibits a diagonal pattern instead. As with the previous example, wormhole records the complete outcome stream the first time we encounter the candidate branch. The next time the inner loop is encountered, however, wormhole does not simply replay the

recorded history. Instead, it attempts to identify correlations using a history fragment comprising bits from the past iteration and the current iteration streams. This is shown in Figure 2(a), where the top row represents the recorded loop history from the previous inner loop invocation, and the second, shorter row shows the history for the current inner loop invocation. The darkened outcome refers to the current execution we are trying to predict. Using LP_{CURR}, wormhole indexes the past iteration history to select a few bits, three in our example, "100". In addition, wormhole can take a few bits from the current iteration history, one in our example, "0", and form the resulting history "0100" upon which to build a correlation with the next outcome. In this case, wormhole learns that the history "0100" leads to outcome "1". This is recorded in a table which uses saturating counters to assign confidence to this prediction.

Figure 2(b) shows how wormhole now correctly predicts the diagonal the next time the inner loop executes. Again it uses three bits from the past inner loop history and one from the current forming again the pattern "0100". This has been found to correlate with a "1" outcome which is correct in this case. Similarly, wormhole can predict all instances of the branch in this case.

Besides diagonals, the wormhole mechanism finds correlations using past iteration history, patterns from the current loop history and the current loop count. For example, it will correctly predict branches that exhibit partial diagonals, or multiple ones. This scheme also predicts correctly those branches that exhibit identical behavior across iterations of the outer loop as in the first example of this section.

IV. ISL-TAGE BASE PREDICTOR

Wormhole is implemented on top of the state-of-the-art ISL-TAGE predictor [20] while re-using some of ISL-TAGE's components to selectively activate the wormhole predictor. As Figure 3 shows, the complete prediction framework comprises four different predictors: 1) a TAGE predictor comprising a bimodal predictor and several tagged components [18], 2) a loop predictor, 3) a statistical corrector, and 4) a wormhole predictor. The first three components comprise the baseline ISL-TAGE. This section reviews the baseline ISL-TAGE predictor [20], while Section V describes the wormhole component and its integration on top of ISL-TAGE.

A. TAGE Predictor

The TAGE predictor consists of a bimodal predictor and several global history indexed tagged tables. Each tagged table uses a different history length and these lengths form a geometric series. The bimodal table captures well-behaved biased branches, whereas the rest of the tables capture exceptions, that is history patterns for events that foil the bimodal component. By using different history lengths and an adaptive table selection mechanism, TAGE effectively identifies correlations in the global history.

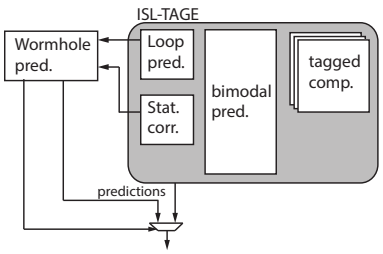


Figure 3: Design overview.

B. Loop Predictor

The loop predictor targets loops that execute a constant number of iterations. It tracks how many times a branch had the same outcome before having the contrary outcome, an indication that the loop exit has been reached. The loop predictor overrides the other components of the branch predictor when the loop has appeared seven consecutive times with the same number of iterations. When a branch finds its corresponding entry in the loop predictor enabled, a global register, LP_{TOTAL}, starts tracking the number of iterations for the loop. As Section V explains, wormhole uses this count to determine: 1) whether a branch is inside a loop, and 2) if so, the expected total iteration count for the loop. Using this information, wormhole decides when to fold the local history into a multidimensional representation.

C. Statistical Corrector

The statistical corrector (SC) targets those branches that TAGE fails to predict accurately but that are statistically biased towards a certain direction. The statistical corrector monitors TAGE's predictions and dynamically decides whether to invert them. SC estimates TAGE's accuracy for the current branch using a set of saturating counter tables. These tables are indexed with the TAGE prediction and a fragment of the global history. The lengths of the fragments are the same as those used in the tagged components of TAGE. Seznec describes the decision and the update processes [21]. As Section V explains, wormhole multi-purposes the same statistical corrector to identify candidate branches for wormhole prediction.

V. WORMHOLE PREDICTOR

The wormhole predictor treats local history bit vectors as multidimensional bit matrices, folding them over iterations of the outer loop(s). This enables wormhole to correlate with previous iterations of both the inner and the outer loops. Wormhole prediction proceeds in four stages: 1) Identify branches that are frequently mispredicted by the base TAGE predictor. 2) Detect the *dimensionality* of the current loop nest, that is how many times the loop iterates. 3) Record

the local branch history, and 4) Learn patterns in the multidimensional local history space. The rest of this section describes these stages.

A. Identifying Problematic Branches

Wormhole specifically targets branch instructions that are problematic for the base ISL-TAGE predictor. To identify such branches, wormhole leverages information in the statistical corrector and the loop predictor. It uses Seznec's original statistical corrector (SC), to identify problematic branches. The existing SC tracks whether TAGE often fails to predict a branch. The SC then overrides TAGE's prediction with the branch's bias if any exists. Wormhole uses only the first part, so that a branch becomes a candidate for wormhole prediction even if it does not exhibit a bias. However, this is not sufficient, the branch must also appear inside a loop. This information is readily available via the loop predictor. If there is an active entry and LP_{TOTAL} is non-zero, the branch is inside a loop.

Once a candidate branch is identified, wormhole allocates an entry in its wormhole prediction table (WPT). WPT entries are ranked; if the WPT is full, the entry with the lowest ranking is selected for replacement. Whenever the statistical corrector deems a branch to be problematic, its entry moves up one spot in the ranking and the statistical corrector continues to identify it as problematic. In this way, the WPT identifies branch instructions that are frequently mispredicted by the base predictor.

B. Detecting Loop Dimensionality

To detect loop dimensionality, that is the expected number of iterations, wormhole leverages information in the loop predictor of ISL-TAGE. As Section IV-B discussed, whenever a branch hits in the loop predictor, a global register (LP_{TOTAL}) is updated. If the loop is currently in progress, then LP_{TOTAL} stores the total number of iterations in the loop. When the loop terminates, LP_{TOTAL} is reset to zero. Thus, at any point in time, LP_{TOTAL} stores the total expected number of iterations for the current innermost loop. The wormhole predictor uses this information to record and fold the history for candidate branches. In general, this mechanism can represent local history bit vectors as multidimensional bit matrices. The current implementation utilizes only two dimensions.

C. Recording the Local History

Once a candidate branch and the loop's dimensionality are identified, wormhole records the local history of the branch. Figure 4 shows the format of the wormhole predictor entry. The length of the local history field determines the maximum history that wormhole can record. Larger predictors use additional history bits to correlate with older iterations. A confidence counter tracks wormhole success and thus whether it should override any other predictors.

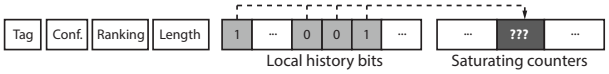


Figure 4: Wormhole predictor entry.

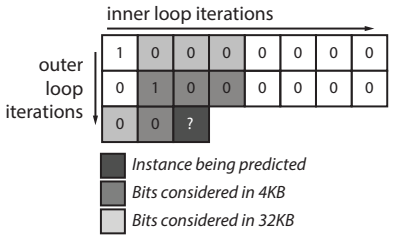


Figure 5: Bits considered by the wormhole predictor

D. Learning 2D Patterns

Wormhole next identifies patterns in the 2D history space comprising both past and current iterations recorded in the entry history for the branch. A history string containing local branch outcomes from the previous iterations of both the inner (horizontal) and outer (vertical) loops is constructed. Figure 5 shows the specific history bits used in this work. The darkest square, labelled as "?", represents the instance of the branch that is being predicted. Depending on the resources dedicated to wormhole, the number of bits used to make a prediction differs. We consider two different configurations: 4KB and 32KB. Section VI-A details these two predictor configurations. In Figure 5, dark grey squares show the history bits the 4KB predictor uses. These bits correspond to locations 0 (most recent), LP_{TOTAL}, $LP_{TOTAL}-1$ and $LP_{TOTAL}-2$ in the local history vector. Light grey squares show the bits that are additionally considered to make a prediction in the 32KB predictor. In this case, these additional bits correspond to the bits 1, $2 \times LP_{TOTAL}$, $2 \times LP_{TOTAL}-1$ and $2 \times LP_{TOTAL}-2$.

The selected local history bits index into a table of saturating counters, which is embedded within the entry as Figure 4 shows; these counters provide the prediction for the branch (the sizes of these counters are shown in Table II). The wormhole prediction is prioritized above the base ISL-TAGE prediction as long as the confidence counter of the entry is positive and the saturating counter for the pattern matches the equation:

$$abs(2 \times sat\_value + 1) >= threshold$$

Where sat_value is the value of the 5-bit saturating counter. A $threshold$ of 16 worked well for the specific workloads. The confidence counter of the corresponding wormhole predictor entry is only updated when its prediction differs from the TAGE prediction. It is increased if the wormhole prediction matched the last outcome of the branch, and decreased otherwise.

Trace name	Total CBr. (dynamic)	Diff. CBr. (static)	ISL-TAGE 4KB MPKI	ISL-TAGE 32KB MPKI	Trace name	Total CBr. (dynamic)	Diff. CBr. (static)	ISL-TAGE 4KB MPKI	ISL-TAGE 32KB MPKI
LONG-00	25.2M	5130	2.521	1.43	**FP-1**	2.23M	460	1.654	1.192
LONG-01	25.3M	89	7.864	7.059	**FP-2**	3.81M	2523	0.869	0.459
LONG-02	22.7M	544	2.058	0.311	**FP-3**	3M	1091	0.015	0.014
LONG-03	16.7M	209	1.339	0.628	**FP-4**	4.87M	2256	0.015	0.014
LONG-04	31.5M	72	9.388	8.746	**FP-5**	2.56M	4536	0.008	0.007
LONG-05	9.4M	129	5.307	4.692	**INT-1**	2.21M	444	6.94	0.137
LONG-06	27.1M	377	0.68	0.606	**INT-2**	1.79M	452	8.577	4.552
LONG-07	23.5M	4080	17.667	8.41	**INT-3**	1.55M	810	11.201	7.307
LONG-08	14.6M	1184	0.718	0.593	**INT-4**	0.89M	556	1.417	0.555
LONG-09	20.5M	810	4.118	3.399	**INT-5**	2.42M	243	0.074	0.059
LONG-10	14.3M	734	2.318	0.632	**MM-1**	4.18M	424	7.585	6.809
LONG-11	16.1M	168	0.815	0.513	**MM-2**	2.87M	1585	10.065	8.739
LONG-12	19.7M	209	11.302	10.939	**MM-3**	3.77M	989	0.066	0.057
LONG-13	27.9M	862	12.611	5.33	**MM-4**	2.07M	681	0.992	0.916
LONG-14	29.5M	25	0.001	0.001	**MM-5**	3.75M	441	5.052	3.518
LONG-15	16.8M	880	1.027	0.274	**SERV-1**	3.66M	10910	5.569	0.783
LONG-16	22M	732	3.227	2.986	**SERV-2**	3.54M	10560	5.97	0.755
LONG-17	14.8M	388	3.351	2.361	**SERV-3**	3.81M	16604	4.596	2.742
LONG-18	19.7M	128	0.005	0.003	**SERV-4**	4.27M	16890	5.344	1.903
LONG-19	14.4M	684	1.349	1.002	**SERV-5**	4.29M	13017	5.513	1.531

Table I: Total number of conditional branches, number of unique conditional branches, MPKI for ISL-TAGE 4KB, and 32KB, for the 40 traces.

E. Wormhole Predictor Table Entry Format

Figure 4 shows a wormhole predictor table entry. The size of each field depends on the predictor configuration. The fields are as follows:

Tag: Identifies the target branch via its PC.

Conf: A saturating counter that tracks how well wormhole is predicting the corresponding branch.

Ranking: Used by the replacement policy.

Length: Expected number of iterations of the inner loop containing the target branch.

History: Local history of the branch.

Saturating counters: A table of counters that provide the direction prediction.

VI. Evaluation

This section demonstrates the potential of wormhole prediction. Section VI-A details the experimental methodology. Section VI-B shows that WISL-TAGE improves accuracy over ISL-TAGE [20]. Section VI-C demonstrates that wormhole offers similar if not better benefits compared to existing side-predictors. Section VI-D considers the interaction of wormhole with the most recently proposed TAGE-SC-L [22] showing that wormhole can improve overall accuracy while complementing TAGE-SC-L's *Local History Based Statistical Correctors*.

To illustrate how wormhole improves accuracy, Section VI-E takes a closer look at *hmmer*, the workload that benefits the most. This analysis considers the effects of the data input and of the compiler and shows that: 1) the phenomenon wormhole exploits persists across data inputs, and 2) the compiler could convert the branch into a conditional move thus eliminating the need for prediction. The latter observation motivates the analysis of Section VI-F that shows the overall impact the compiler has on branch prediction accuracy reaffirming that using optimizations such as conditional moves is not free of trade offs. Finally, Section VI-G shows the storage requirements of the various previously considered predictors.

A. Methodology

For the sake of reproducibility, we use the experimental framework from the 4th Branch Prediction Competition [1]. The framework is trace-driven and includes 40 different traces. The first 20 traces are from the SPEC CPU 2006 benchmarks [9] and are each 150 million instructions long, while the next 20 are 30 million instructions long and are from four different application domains, namely: floating point, integer, multimedia, and server. These traces include system and user activity; Table I presents some of their characteristics. In order, the table's columns report the name of the trace, the total number of dynamic conditional branches, the number of unique conditional branches (static), and the number of mispredictions per kilo instruction for the 4KB and 32KB ISL-TAGE baseline predictors.

1) Predictor configurations: The storage dedicated to the branch predictor in an aggressive out-of-order processor is on the order of 32KB [23]. At the same time, a smaller size of 4KB is closer to what is found in less-aggressive and power/area-constrained modern processors. To capture both these target applications, we consider two different sizes for our base predictors throughout the paper: 32KB and 4KB.

The base predictor, ISL-TAGE comprises several components: 1) the bimodal predictor features 2^{13} and 2^{14} entries

	4KB	32KB
Tag	18	18
Confidence	4	4
Sat. counters	5 (x 16)	5 (x 256)
Ranking	3	3
History vector	101	257
History length	7	9

Table II: Components of each wormhole predictor entry (all sizes in bits).

for the 4KB and 32KB designs, with 2 bits per entry. 2) There are 15 TAGE components whose number of entries is assigned from the best performing design in prior work [20]. 3) The loop predictor has 64 entries, is 4-way skewed associative and can track loops with up to 2^{10} iterations. 4) The statistical corrector features 64 and 256 entries for the 4KB and 32KB designs respectively. Each entry is 5 bits.

Table II shows the wormhole predictor configurations. Each entry has a table with 16 or 256 5-bit saturating counters to predict the different patterns, a 4-bit confidence counter, a 3-bit ranking counter, a 101- or 257-bit vector to store the local history within loop iterations, and a 7- or 9-bit counter to store iteration count that is considered for the corresponding branch. Section VI-G analyzes the hardware storage needed to implement the predictors.

B. Comparison with ISL-TAGE

Figure 6 shows the reduction in MPKI for WISL-TAGE with respect to 4KB and 32KB base ISL-TAGE predictors. WISL-TAGE improves the misprediction rate of 16 and 35 out of 40 traces for the 4KB and the 32KB base predictors, respectively. On average, WISL-TAGE reduces the MPKI by 2.53% and 3.15% for the 4KB and 32KB ISL-TAGE base predictors. Considering only the four top benchmarks, WISL-TAGE reduces the MPKI of the ISL-TAGE base predictors by 22% and 20%, respectively for the 4KB and 32KB configurations.

C. Putting Wormhole Accuracy in Perspective: A Comparison With Other Side-Predictors

This section analyzes the contribution of two existing side-predictors: the loop predictor and the statistical corrector, to the overall accuracy of ISL-TAGE. Figures 7a and 7b show the reduction in mispredictions with respect to a 4KB and 32KB TAGE predictor. The first three bars show reductions in mispredictions when a loop predictor, a statistical corrector predictor, and both side-predictors are added on top of the base predictor. The fourth bar shows reductions when the wormhole predictor is added on top of the base predictor and the other two side-predictors.

In the 4KB case (Figure 7a), the loop predictor reduces MPKI by over 10% in four benchmarks (LONG-18, LONG-19, FP-5, INT-5, and MM-4), while the statistical corrector achieves a significant reduction in MPKI for only one benchmark (LONG-18). The wormhole predictor significantly improves the accuracy of the predictions in four of the benchmarks (LONG-12,[2] LONG-18, FP-1, and MM-4), and slightly improves the misprediction ratio of almost all other applications. On average, MPKI reductions are 5.8%, 1.1%, 6.9%, and 9.6% for the loop, statistical corrector, ISL-TAGE, and WISL-TAGE. For the four top benchmarks, WISL-TAGE reduces the MPKI by 40% over the TAGE base predictor. The results in the 32KB case are similar with average MPKI reductions 4.9%, 1.5%, 6.4%, and 8.31% for the loop, statistical corrector, ISL-TAGE, and WISL-TAGE. For the four top benchmarks, WISL-TAGE reduces the MPKI by 38% over the TAGE base predictor. The benefits brought by the wormhole predictor are comparable to those brought by the loop predictor. As Section VI-G shows, all three side-predictors have similar hardware costs.

D. TAGE-SC-L

This section considers the interaction of wormhole with the recently proposed TAGE-SC-L [22], winner of the 4th Branch Prediction Championship [1]. TAGE-SC-L is an improvement of ISL-TAGE. It simplifies the loop predictor and the statistical corrector of its predecesor; they use a smaller fraction of the hardware budget so more resources can be devoted to the TAGE components. We analyze two aspects of the TAGE-SC-L predictor: 1) The accuracy of its components that use local branch histories compared to the wormhole predictor. 2) The accuracy of the wormhole predictor when it is incorporated into TAGE-SC-L.

1) Accuracy of the Local History Based Components: The 32KB version of TAGE-SC-L [22] includes local history based components (*LHCs*). These components, similar to those previously presented by Seznec [21], capture branches whose behavior is correlated with their own local history but that could not be properly predicted by global history based predictors. Branches targeted by wormhole fall into this category.

Figure 8 shows the accuracy of four predictors for each workload. From left to right the bars are for 1) the TAGE-SC-L without the LHCs, 2) TAGE-SC-L without the LHCs but with a wormhole component, 3) the original 32KB TAGE-SC-L, and TAGE-SC-L with a wormhole component. All results are relative to the MPKI of the ISL-TAGE base predictor. For readability, the graph omits seven of the benchmarks that have less than 0.1 MPKI and whose MPKI variation is minimal. Comparing the first and third bars shows that the LHCs result in small improvements in several workloads. The use of the *LHCs* reduces the MPKI of TAGE-SC-L by 3.44% on average. Comparing the second and third bars shows that the benefit of wormhole is more concentrated, with significant improvements for LONG-12 and MM-4, leading to an improvement in MPKI over the

[2]LONG-12 corresponds to *hmmer*.

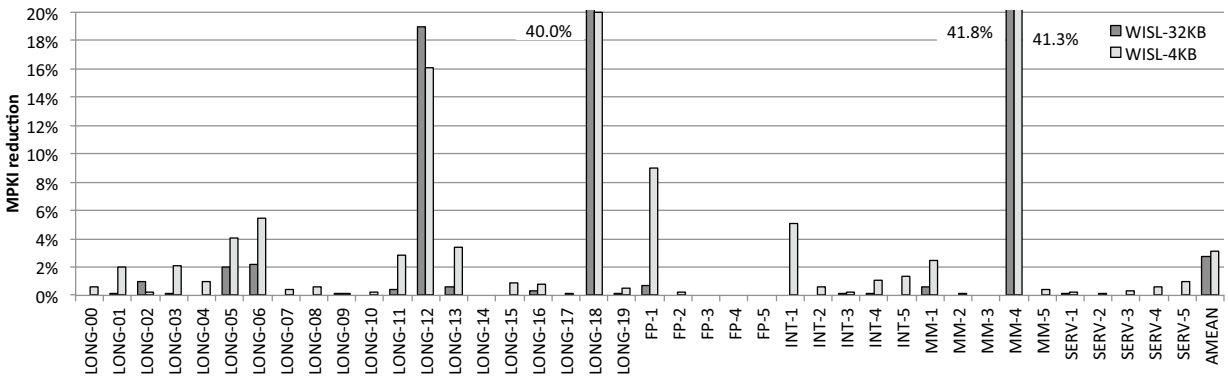


Figure 6: MPKI reductions with respect to ISL-TAGE for the 40 traces, for 4KB and 32KB base predictors.

base predictor of 2.77%. Both types of local predictors can be combined (fourth bar, *SC-L-32KB-original+WH*) and their individual benefits mostly remain, reducing the average MPKI by 5.93% compared to TAGE-SC-L without *LHCs*.

2) Accuracy of the Wormhole Predictor on Top of TAGE-SC-L: The last two bars of each group in Figure 8 show reduction in mispredictions of TAGE-SC-L with and without a wormhole predictor (*SCL-WH* in the figure), with respect to ISL-TAGE. For some workloads the TAGE-SC-L-based is less accurate than ISL-TAGE as it devotes less storage to the statistical corrector. However, the differences are small in absolute terms (see Table I). The wormhole predictor improves the misprediction rate of eight out of 40 traces for the 32KB base predictor. On average, the wormhole predictor reduces the MPKI of the 32KB TAGE-SC-L predictor by 2.6%.

E. A Real World Example: hmmer

The code in Fig. 9(a) shows a fragment of *hmmer* [6] from SPEC CPU 2006 [9]. More precisely, it shows a fragment of the *P7Viterbi* function (some auxiliary variables used as aliases in the orginal code are omitted for clarity). This function is responsible for 95% of the application's execution time. *Branch 1* represents 10.5% of the total dynamically executed branches, and causes 42% of the mispredictions when a 32KB ISL-TAGE branch predictor is used.

Analyzing the code in Figure 9(a) shows that the outcome of *Branch 1* depends on the values of *imx[i-1][k]* and *imx[i][k]*, where the *imx[i][k]* value depends on the value of *mmx* from the previous iteration of the outer loop. The remainder of the code is immaterial to this discussion; the important point to note is that the outcome of Branch 1 depends on data carried across iterations of the outer loop.

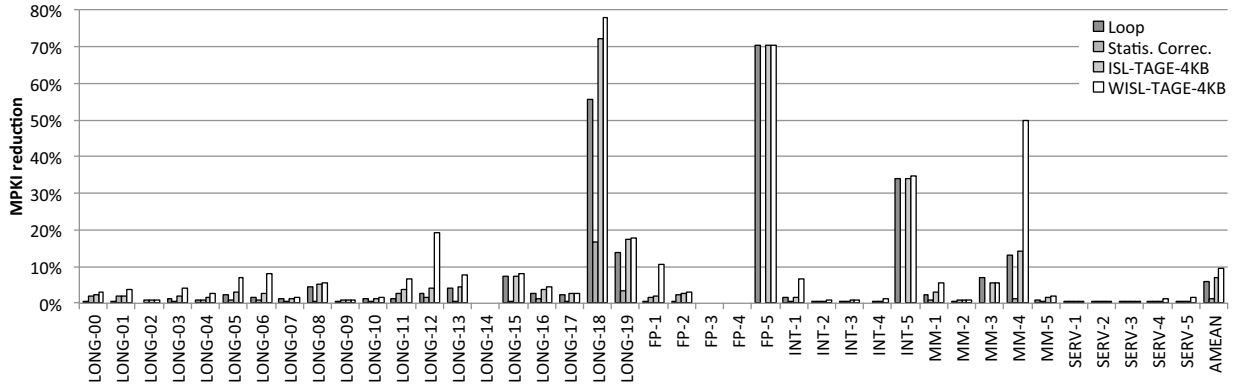
Figures 9(b) and (c) show correct predictions of *Branch 1* for a subset of iterations of Loops 1 and 2, made by ISL-TAGE and WISL-TAGE, both 4KB. The elements in each row represent iterations of Loop 2, from k=1 to k=79 (for clarity the figure omits the remaining iterations). Each row represents a different iteration of Loop 1 and the figure shows ithe terations from i=927 to i=942. Blank spaces indicate mispredicted branches. Figure 9(c) highlights in grey the outcomes of Branch 1 that are correctly predicted by WISL-TAGE but not by ISL-TAGE. WISL-TAGE is able to detect most of the columns in the pattern and some diagonals that ISL-TAGE is unable to predict. Section VI-E2 shows that this behavior persists even with different inputs.

1) Conditional Moves: The code of *Hmmer* in Figure 9(a) shows that the body of Branch 1 consists only of an assignment operation. Although this branch is present in the traces used in the 4th Branch Prediction Competition [1], when a more modern compiler is used this type of branch will likely be converted to a *Conditional Move (CMOV)* instruction. Figure 10 shows the assembler code (x86) generated by two different versions of GCC. GCC 4.0 (Figure 10(a)) generates a conditional branch while GCC 4.6 (Figure 10(b)) generates instead a conditional move instruction *cmovge*.

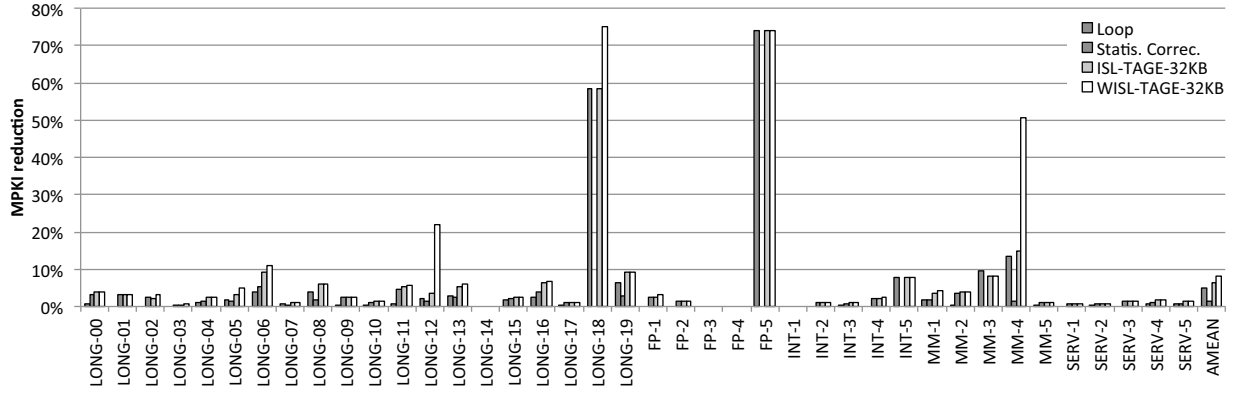
2) Sensitivity to the Input Dataset for hmmer: This section shows that behavior the wormhole predictor exploits is not happenstance but persists with other input datasets for *hmmer*. The *hmmer* application in SPEC CPU 2006 uses the (*nph3*) query in the (*swiss41*) database. This section reports the results of running *hmmer* using two other queries, *globins4* and *fn3* and with a newer and bigger protein database *Swiss-Prot [26]*. Figure 11 shows the MPKI for 4KB and 32KB ISL-TAGE and WISL-TAGE predictors. In the case of *globins4*, the wormhole predictor reduces MPKI by around 45% and 33% for the 32KB and 4KB predictors. While, in the case of *fn3*, the corresponding MPKI reductions with the wormhole predictor are 32% and 25%. Although the key branch in *hmmer* is data dependent, the patterns repeat across iterations of the outer-loops enabling wormhole to remain effective across different inputs.

F. Interaction with the Compiler

Leveraging the traces from the 4th Branch Prediction Championship does not afford us the ability to recompile the applications, which are not publicly released. To study the impact of the compiler on the performance of the



(a) 4KB base predictors



(b) 32KB base predictors

Figure 7: MPKI reductions for each side-predictor: Loop predictor, statistical corrector, ISL-TAGE (Loop+SC), and WISL-TAGE (Loop+SC+WH).

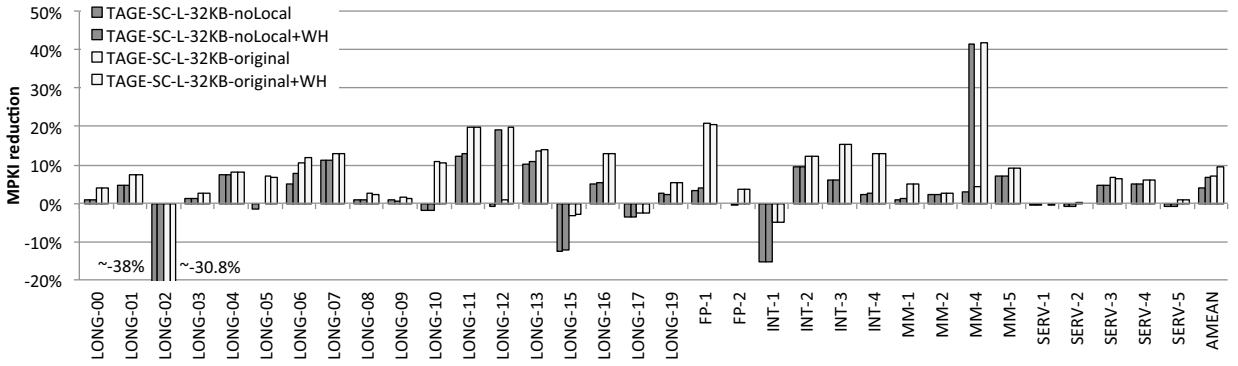


Figure 8: MPKI variation for TAGE-SC-L with and without wormhole compared to the 32KB ISL-TAGE baseline for 33 traces.

Figure 9: (a) Fragment of the P7Viterbi() function. (b) ISL-TAGE 4KB correct predictions of Branch 1. (c) WISL-TAGE 4KB correct predictions of Branch 1.

456.hmmer/run/build-gcc40/fast_algorithms.c:147

```
mov   -0x34(%ebp),%edi
mov   -0x4(%edi),%eax
mov   -0x48(%ebp),%edi
add   -0x4(%edi,%ecx,1),%eax
cmp   %eax,%edx
jge   80515e5 <P7Viterbi+0x285>
mov   %eax,-0x4(%ebx)
```

(a)

456.hmmer/run/build-gcc46/fast_algorithms.c:147

```
mov   (%edi,%eax,4),%esi
mov   0x78(%esp),%ebp
add   0x0(%ebp,%eax,4),%esi
cmp   %ecx,%esi
cmovge %esi,%ecx
mov   0x3c(%esp),%ebp
mov   %ecx,0x0(%ebp,%eax,4)
```

(b)

Figure 10: Assembler code (x86) generated for Branch 1. (a) Using GCC 4.0. (b) Using GCC 4.6.

wormhole predictor, we have compiled applications from the SPEC CPU 2006 suite using the GCC versions 4.0 and 4.6. Figures 12a and 12b compare the MPKI of ISL-TAGE and WISL-TAGE for 4KB and 32KB predictors. The MPKI is on average ∼10% lower when a modern compiler is used and the relative differences between ISL- and WISL-TAGE are smaller for both storage budgets.

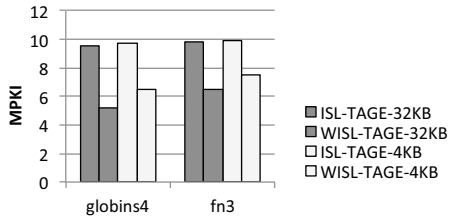


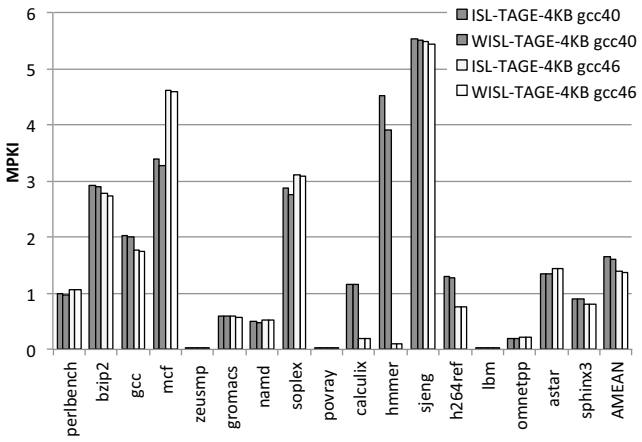
Figure 11: MPKI for ISL- and WISL-TAGE with alternative dataset inputs for hmmer.

The MPKI reduction observed with GCC 4.6 is primarily due to *calculix* and *hmmer*. These MPKI reductions are 4.8% and 0.5% for GCC 4.0 and 4.6 with 32KB predictors and 3.6% and 1.0% for GCC 4.0 and 4.6 with 4KB predictors. Besides these two programs, using a more recent compiler does not always improve accuracy. However, applying wormhole yielded significant benefits for some workloads without hurting accuracy for the remaining ones.

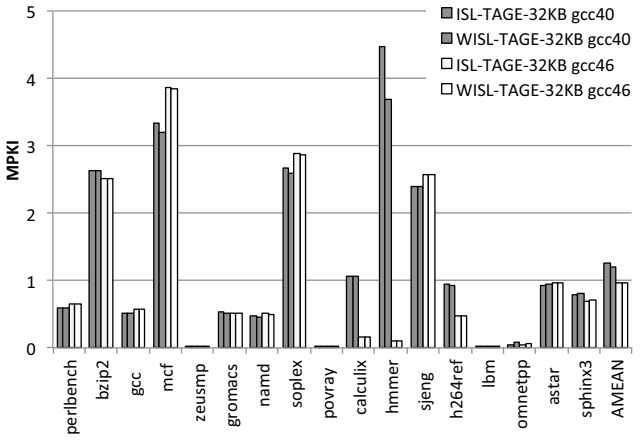
Finally, the compiler chose to use a conditional move because the body of the if statement is a simple assignment operation. A more complex if clause could not be optimized away by the compiler and it would still benefit from wormhole prediction.

G. Storage

Section IV presented all the components of the predictors used in this work and their corresponding configurations. Table III shows the storage used by the different components



(a) 4KB predictors.



(b) 32KB predictors.

Figure 12: MPKI comparison using GCC 4.0 and GCC 4.6.

	4KB	32KB
Statistical corrector, size (bytes)	96	384
Loop predictor, size (bytes)	376	376
TAGE predictor, size (bytes)	3048	29952
ISL-TAGE extra, size (bytes)	524	524
Wormhole predictor, size (bytes)	134	1375
Total size (bytes)	4178	32611

Table III: Storage of the different predictors.

of our design. The total storage employed by the WISL-TAGE design is 4,178 and 32,611 bytes for the 4KB and 32KB budgets. The wormhole predictor requires 1065 and 10997 bits, yielding an overhead of 3.29% and 4.4%, for the 4KB and 32KB configurations. The wormhole predictor is a simple hardware structure with modest overhead that yields a substantial reduction in MPKI for several key workloads and performance improvements across all workloads.

VII. Conclusions

Commercial architectures continue to strive for increased branch prediction accuracy. Not only does branch prediction accuracy can greatly improve performance, it is imperative for energy-efficient design; fetching and executing wrong-path instructions wastes significant energy. Recent advances in branch prediction research were possible by introducing small side predictors that capture certain branch behaviors that larger, general-purpose structures fail to predict accurately. In this vein, this work proposed the wormhole predictor. Wormhole can predict, using multidimensional local histories, inner-loop branches that exhibit correlations across iterations of the outer loops. Wormhole essentially folds the inter-iteration history into a matrix enabling it to easily observe cross-iteration patterns. By doing so, wormhole yielded MPKI reductions of 20% to 22% for four of the applications studied compared to a state-of-the-art 4KB and 32KB ISL-TAGE predictor. Side-predictors are not intended to improve all branches; on average wormhole reduces MPKI by 2.53% and 3.15% for the 40 applications considered. Yet, we believe that substantial gains on 10% of the applications studied warrant dedicating the small amount of silicon (3.3% and 4.4% increase over the baseline predictor) to the wormhole predictor.

VIII. Acknowledgments

The authors would like to thank the anonymous reviewers for their feedback and the members of the computer architecture research group in the University of Toronto. This work was supported by the Natural Sciences and Engineering Research Council of Canada via Discovery, Discovery Accelerator Supplement, and Strategic grants, Qualcomm, the Canadian Foundation for Innovation, the Ontario Research Fund and a Bell Graduate Scholarship.

References

[1] "JWAC-4: Championship branch prediction," 2014. [Online]. Available: http://www.jilp.org/cbp2014/

[2] J. Albericio, J. San Miguel, N. Enright Jerger, and A. Moshovos, "Wormhole branch prediction using multi-dimensional histories," *JWAC-4: Championship Branch Prediction*, 2014.

[3] J. Burkardt, "Implementation of the Jacobi method." http://people.sc.fsu.edu/~jburkardt/cpp_src/jacobi/jacobi.cpp. [Online]. Available: http://people.sc.fsu.edu/~jburkardt/cpp_src/jacobi/jacobi.cpp

[4] A. N. Eden and T. Mudge, "The YAGS branch predictor," in *Proceedings of the 31st Annual International Symposium on Microarchitecture*, 1998.

[5] M. Evers, P.-Y. Chang, and Y. Patt, "Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches," in *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, 1996.

[6] R. D. Finn, J. Clements, and S. R. Eddy, "Hmmer web server: interactive sequence similarity searching." *Nucleic Acids Research*, vol. 39, no. Web-Server-Issue, pp. 29–37, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/nar/nar39.html#FinnCE11

[7] H. Gao, Y. Ma, M. Dimitrov, and H. Zhou, "Address-branch correlation: A novel locality for long-latency hard-to-predict branches," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2008, pp. 74–85.

[8] H. Gao and H. Zhou, "Adaptive information processing: An effective way to improve perceptron predictors," *Journal of Instruction Level Parallelism*, vol. 7, April 2005.

[9] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006. [Online]. Available: http://doi.acm.org/10.1145/1186736.1186737

[10] Y. Ishii, K. Kuroyanagia, T. Sawada, M. Inaba, and K. Hiraki, "Revisiting local history for improving fused two-level branch predictor," in *Proceedings of the 3rd Championship on Branch Prediction*, 2011.

[11] D. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings of the Seventh International Symposium on High Performance Computer Architecture*, 2001.

[12] D. A. Jiménez, "Oh-snap: Optimized hybrid scaled neural analog predictor," in *Proceedings of the 3rd Championship on Branch Prediction*, 2011.

[13] S. McFarling, "Combining branch predictors," TN 36, DEC WRL, Tech. Rep., June 1993.

[14] P. Michaud, A. Seznec, and R. Uhlig, "Trading conflict and capacity aliasing in conditional branch predictors," in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997.

[15] P. Michaud and A. Seznec, "Pushing the branch predictability limits with the multi-potage+sc predictor," *JWAC-4: Championship Branch Prediction*, 2014.

[16] S. Pan, K. So, and J. Rahmeh, "Improving the accuracy of dynamic branch prediction using branch correlation," in *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1992.

[17] S. M. F. Rahman, Z. Wang, and D. A. Jiménez, "Studying microarchitectural structures with object code reordering," in *Proceedings of the 2009 Workshop on Binary Instrumentation and Applications (WBIA)*, 2009.

[18] A. Seznec and P. Michaud, "A case for (partially) tagged geometric history length branch prediction," *Journal of Instruction Level Parallelism*, 2006.

[19] A. Seznec, "The L-TAGE branch predictor," *Journal of Instruction Level Parallelism*, vol. 9, May 2007.

[20] ——, "A 64 Kbytes ISL-TAGE branch predictor," *JWAC-2 : Championship Branch Prediction*, 2011.

[21] ——, "A New Case for the TAGE Branch Predictor," in *The 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, Ed., Dec. 2011. [Online]. Available: http://hal.inria.fr/hal-00639193

[22] ——, "TAGE-SC-L branch predictors," *JWAC-4 : Championship Branch Prediction*, 2014.

[23] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides, "Design tradeoffs for the Alpha EV8 conditional branch predictor," in *Proceedings of the 29th Annual International Symposium on Computer Architecture*. IEEE Computer Society, 2002, pp. 295–306. [Online]. Available: http://dl.acm.org/citation.cfm?id=545215.545249

[24] J. Smith, "A study of branch prediction strategies," in *Proceedings of the International Symposium on Computer Architecture*, 1981.

[25] E. Sprangle, R. Chappell, M. Alsup, and Y. Patt, "The agree predictor: A mechanism for reducing negative branch history interference," in *Proc. of the 24th Annual International Symposium on Computer Architecture*, 1995.

[26] The UniProt Consortium, "Activities at the universal protein resource (uniprot)," *Nucleic Acids Res*, vol. 42, pp. D191–D198, 2014.

[27] T.-Y. Yeh and Y. Patt, "Two-level adaptive branch prediction," in *Proceedings of the 24th International Symposium on Microarchitecture*, 1991.